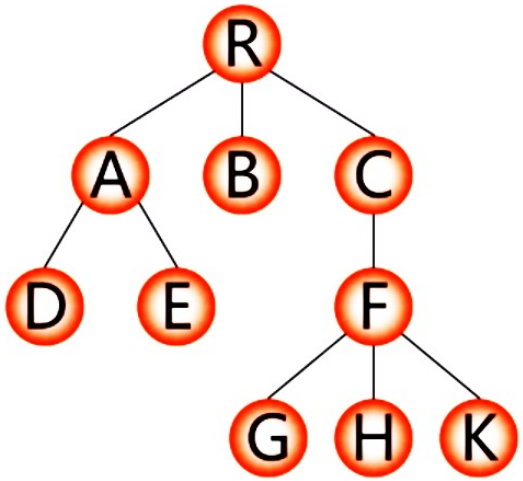
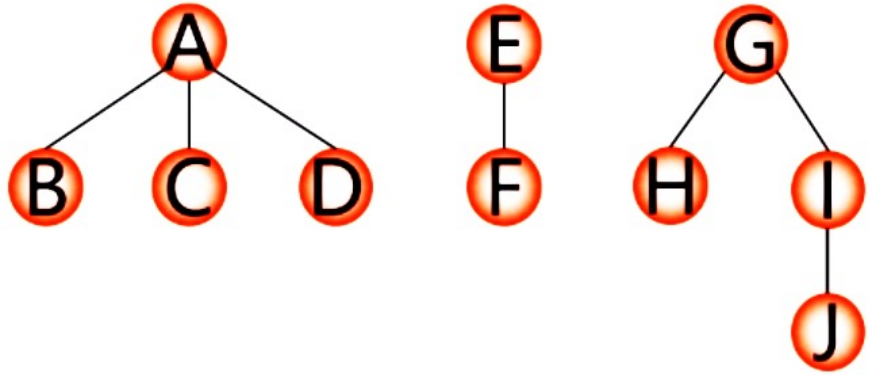
第6章 树和二叉树

6.5 树和森林

树



森林



森林：是m (m≥0) 棵互不相交的树的集合。

树 (Tree) 是n(n≥0)个结点的有限集。若n = 0，称为空树；

若n > 0，(1) 有且仅有一个特定的称为根(Root)的结点；

(2) 其余结点可分为 m (m≥0) 个互不相交的有限集 T1，T2, T3, ..., Tm。

6.5 树的存储结构

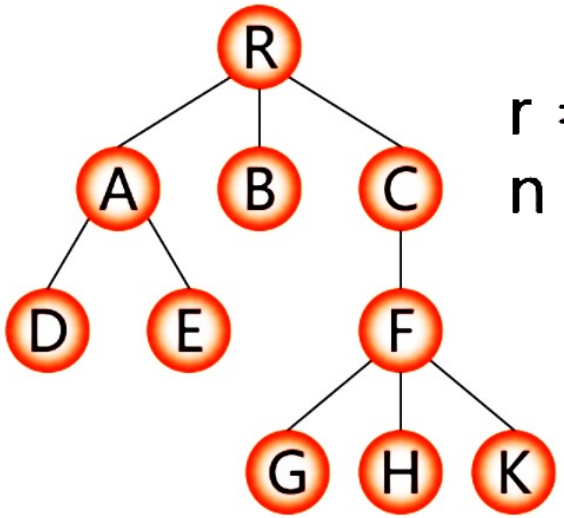
1 双亲表示法

实现：定义结构数组

存放树的结点，

每个结点含两

个域：



r = 0
n = 10

数组下标	data	parent
0	R	-1
1	A	0
2	B	0
3	C	0
4	D	1
5	E	1
6	F	3
7	G	6
8	H	6
9	K	6

- 数据域：存放结点本身信息。
- 双亲域：指示本结点的双亲结点在数组中的位置。

特点：找双亲容易，找孩子难。

6.5 树的存储结构

C语言的类型描述

```
typedef struct PTNode {
   TElemType  data;
   int    parent;   // 双亲位置域
} PTNode;
```

结点结构:

data	parent

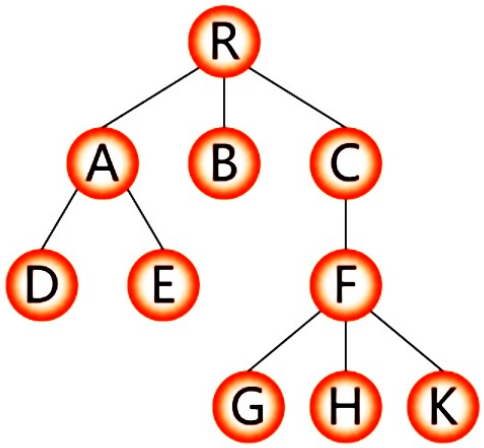
树结构:

```
#define MAX_TREE_SIZE  100
typedef struct {
    PTNode  nodes[MAX_TREE_SIZE];
    int    r, n;     // 根结点的位置和结点个数
  } PTree;
```

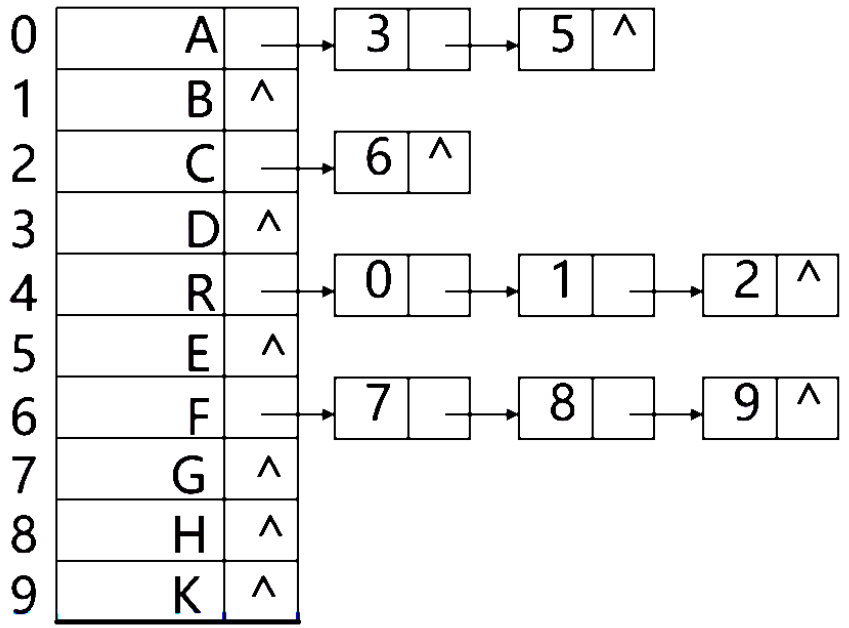
数组下标		data	parent
r = 0	0	R	-1
n = 10	1	A	0
	2	B	0
	3	C	0
	4	D	1
	5	E	1
	6	F	3
	7	G	6
	8	H	6
	9	K	6

6.5 树的存储结构——孩子链表

把每个结点的孩子结点排列起来，看成是一个线性表，用单链表存储，则n个结点有n个孩子链表（叶子的孩子链表为空表）。而n个头指针又组成一个线性表，用顺序表（含n个元素的结构数组）存储。



r = 4
n = 10



6.5 树的存储结构——孩子链表

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

孩子结点结构：

child	next

双亲结点结构：

data	firstchild

```
typedef struct CTNode {
    int          child;
    struct CTNode  *next;
} *ChildPtr;
```

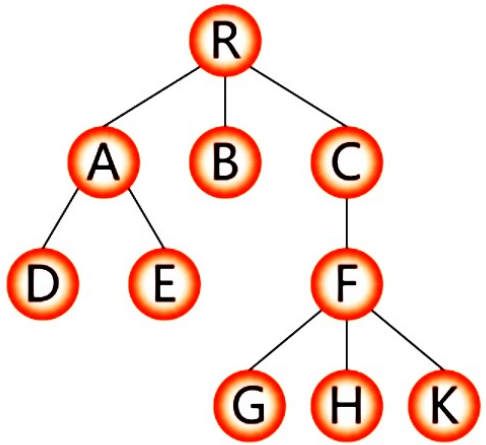
```
typedef struct {
    TElemType    data;
    ChildPtr  firstchild;
                    // 孩子链表头指针
} CTBox;
```

树结构：

```
typedef struct {
    CTBox  nodes[MAX_TREE_SIZE];
    int    n, r;   // 结点数和根结点的位置
} CTree;
```

6.5 树的存储结构——孩子链表

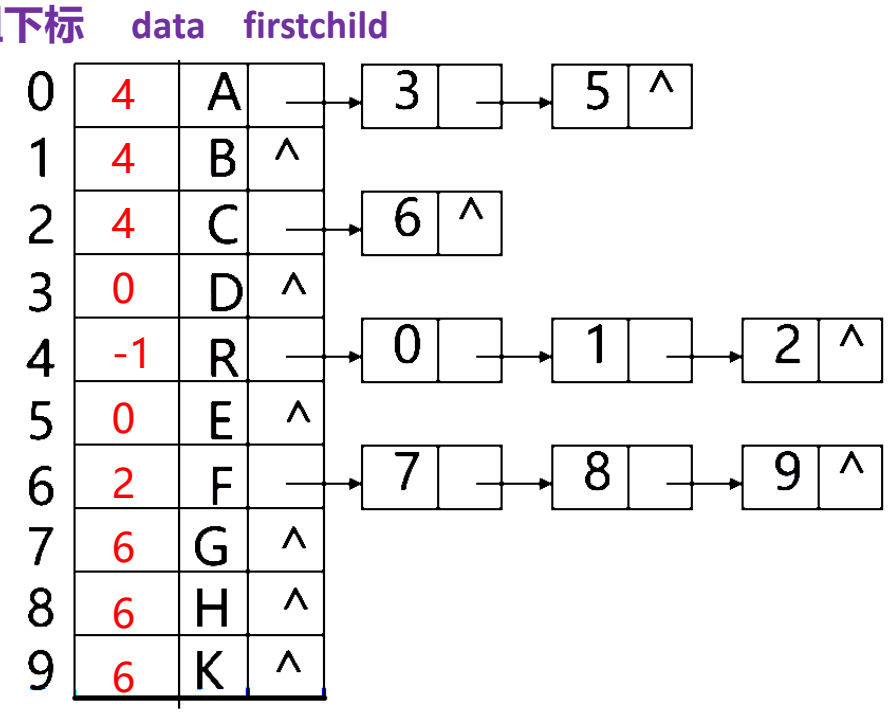
把每个结点的孩子结点排列起来，看成是一个线性表，用单链表存储，则n个结点有n个孩子链表（叶子的孩子链表为空表）。而n个头指针又组成一个线性表，用顺序表（含n个元素的结构数组）存储。



特点：找孩子容易，找双亲难。

r = 4

n = 10



6.5 树的存储结构

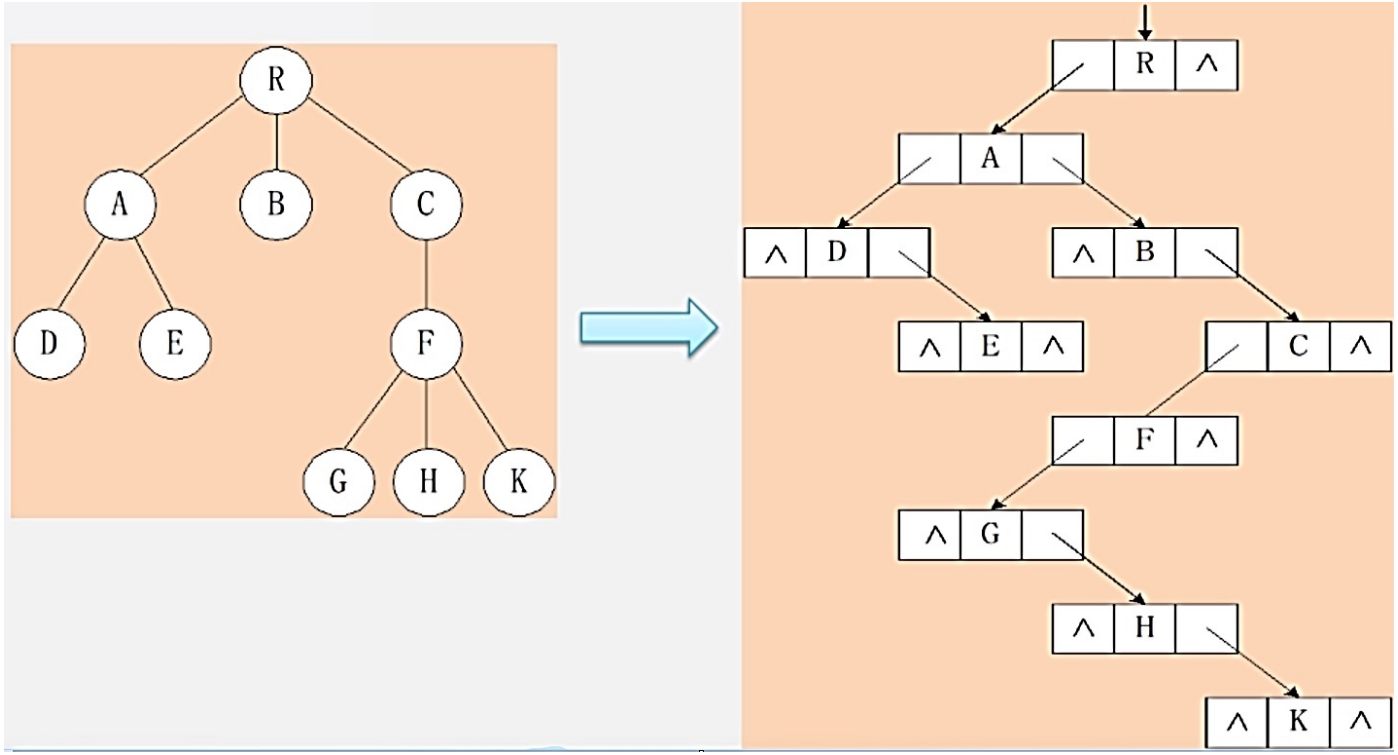
3. 孩子兄弟表示法（二叉树表示法，二叉链表表示法）

实现：用二叉链表做树的存储结构，链表中每个结点的两个指针域分别指向其第一个孩子结点和下一个兄弟结点

```
typedef struct CSNode{
  ElemType          data;
  struct CSNode     *firstchild, *nextsibling;
}CSNode,*CSTree;
```

6.5 树的存储结构——二叉链表表示法

3. 孩子兄弟表示法（二叉树表示法，二叉链表表示法）

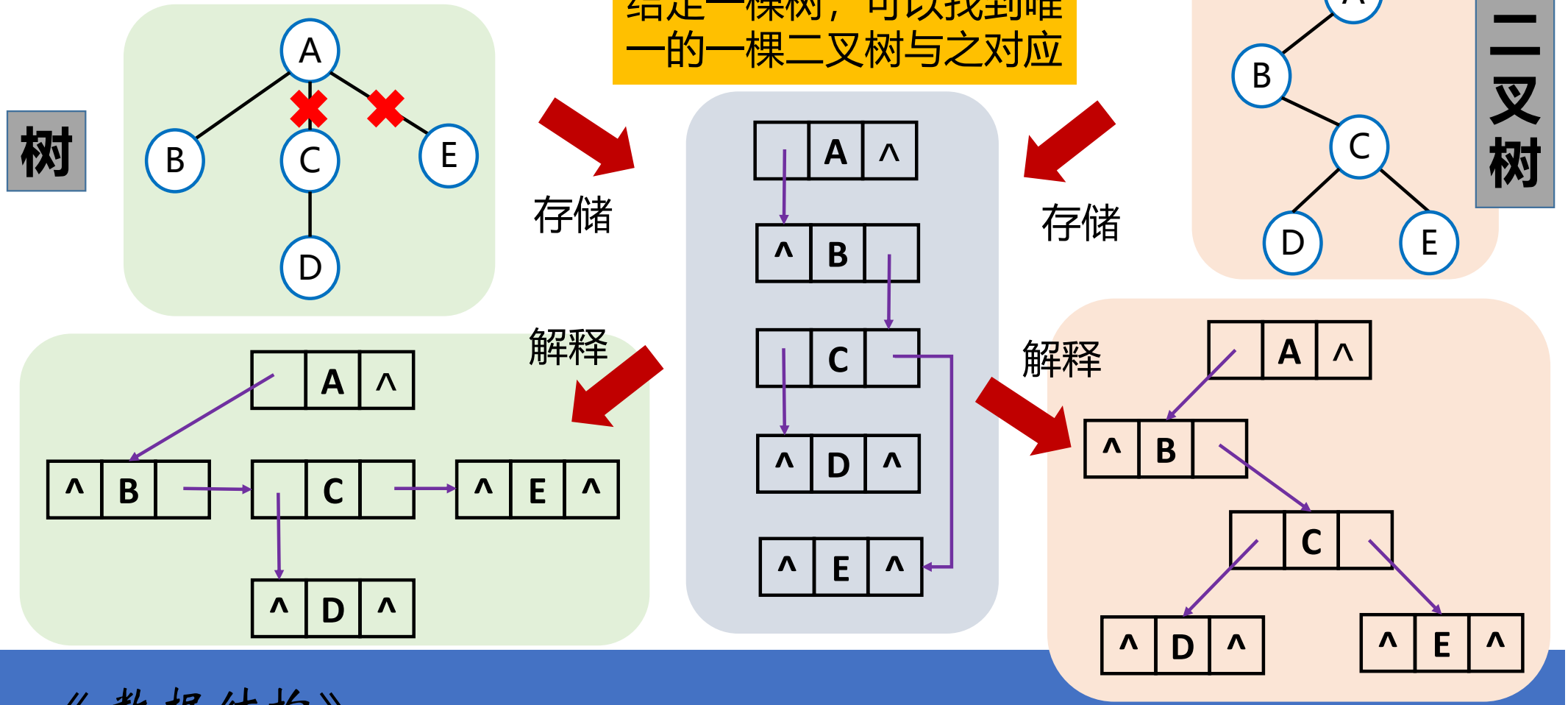


6.5 树与二叉树的转换

- 将树转化为二叉树进行处理，利用二叉树的算法来实现对树的操作。
- 由于树和二叉树都可以用二叉链表做存储结构，则以二叉链表做媒介可以导出树与二叉树之间的一个对应关系。

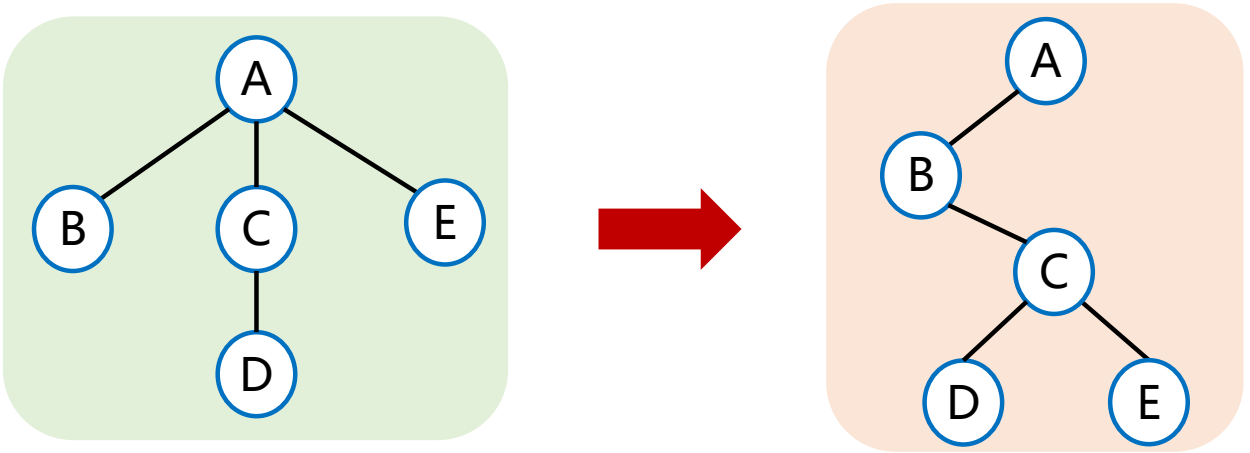
6.5 树与二叉树的转换

给定一棵树，可以找到唯一的一棵二叉树与之对应

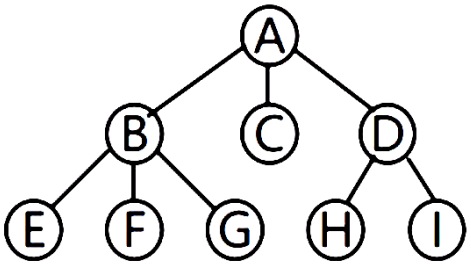


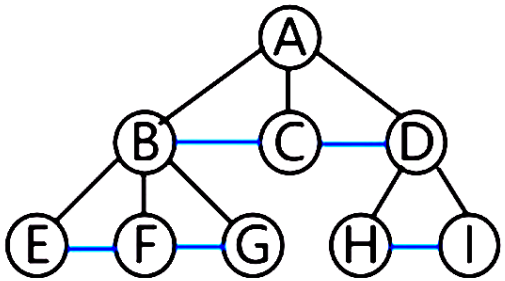
6.5 树与二叉树的转换——将树转换成二叉树

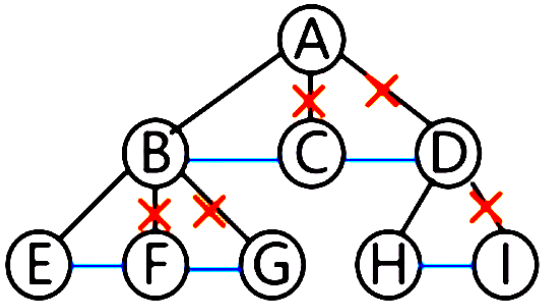
1. 加线：在兄弟之间加一连线
2. 抹线：对每个结点，除了其左孩子外，去除其与其余孩子之间的关系
3. 旋转：以树的根结点为轴心，将整棵树顺时针转45°

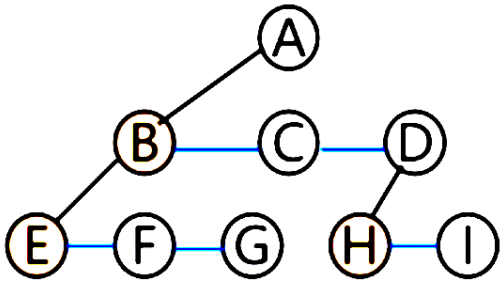


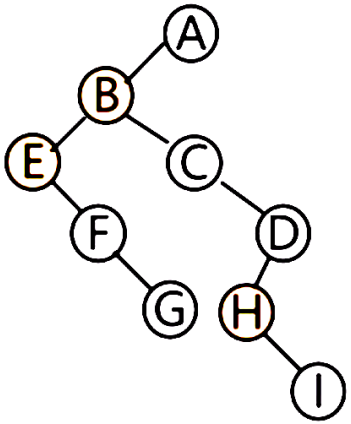
例：将树转换成二叉树





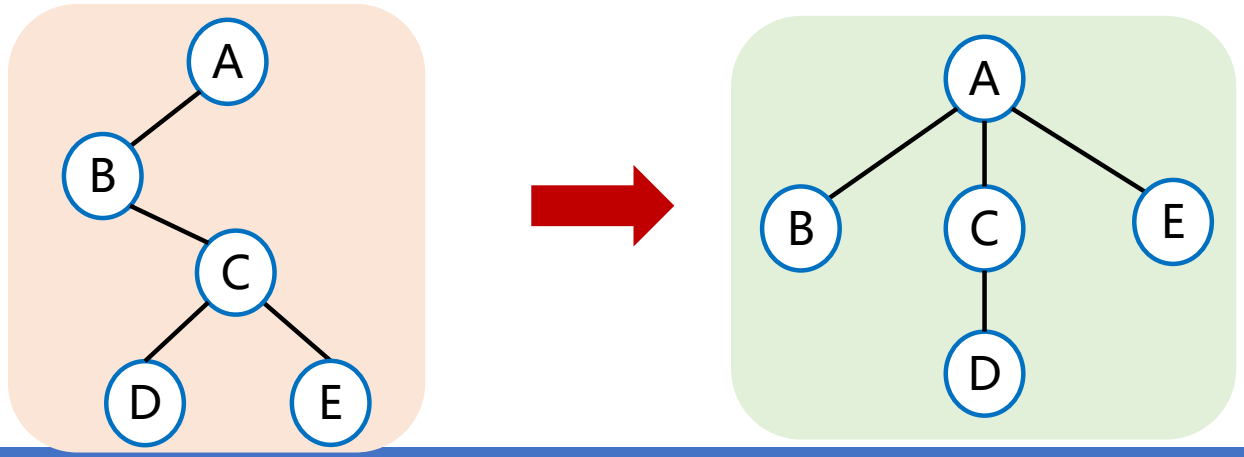




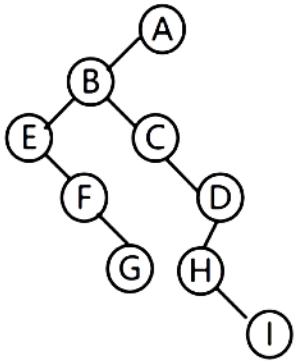


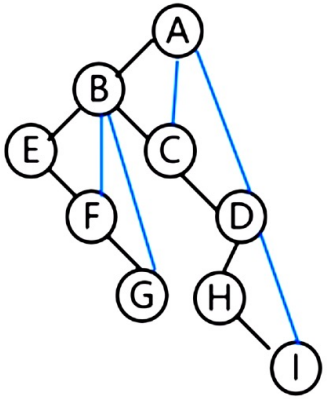
6.5 树与二叉树的转换——将二叉树转换成树

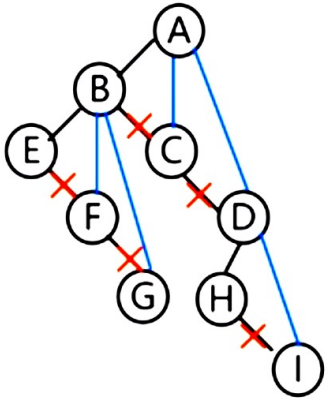
1. 加线：若p结点是双亲结点的左孩子，则将p的右孩子，右孩子的右孩子……沿分支找到的所有右孩子，都与p的双亲用线连起来
2. 抹线：抹掉原二叉树中双亲与右孩子之间的连线
3. 调整：将结点按层次排列，形成树结构

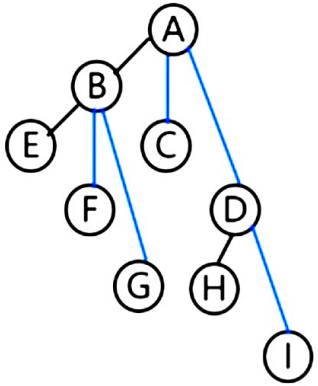


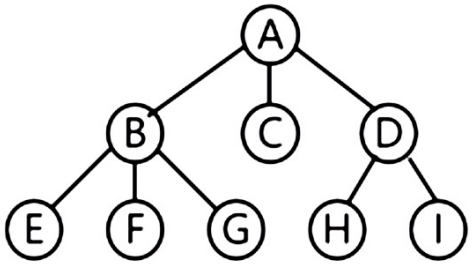
例：将二叉树转换成树









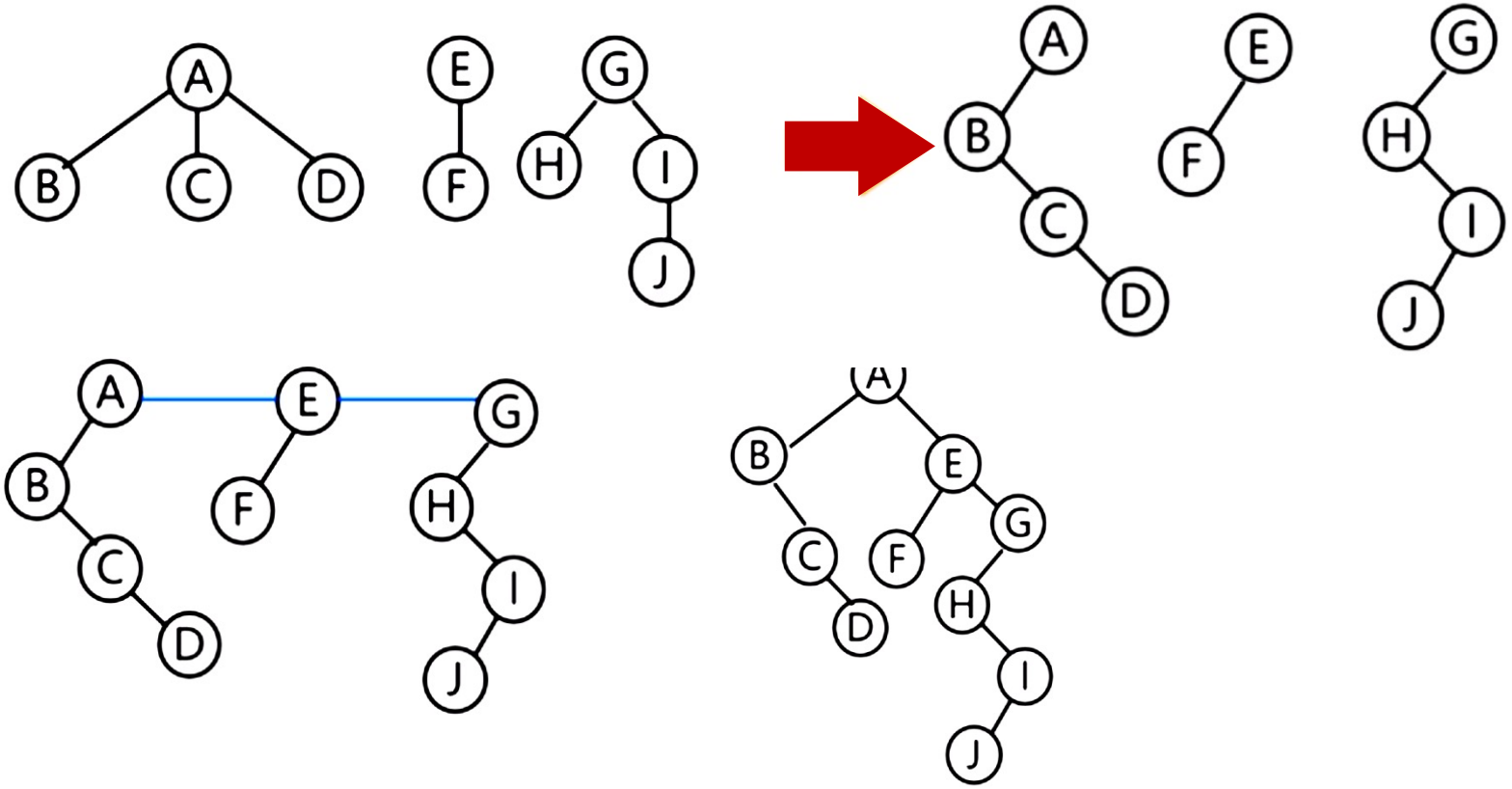


6.5 森林与二叉树的转化

- 森林转换成二叉树（二叉树与多棵树之间的关系）

1. 将各棵树分别转换成二叉树
2. 将每棵树的根结点用线相连
3. 以第一棵树根结点为二叉树的根，再以根结点为轴心，顺时针旋转，构成二叉树型结构

例：森林转换成二叉树

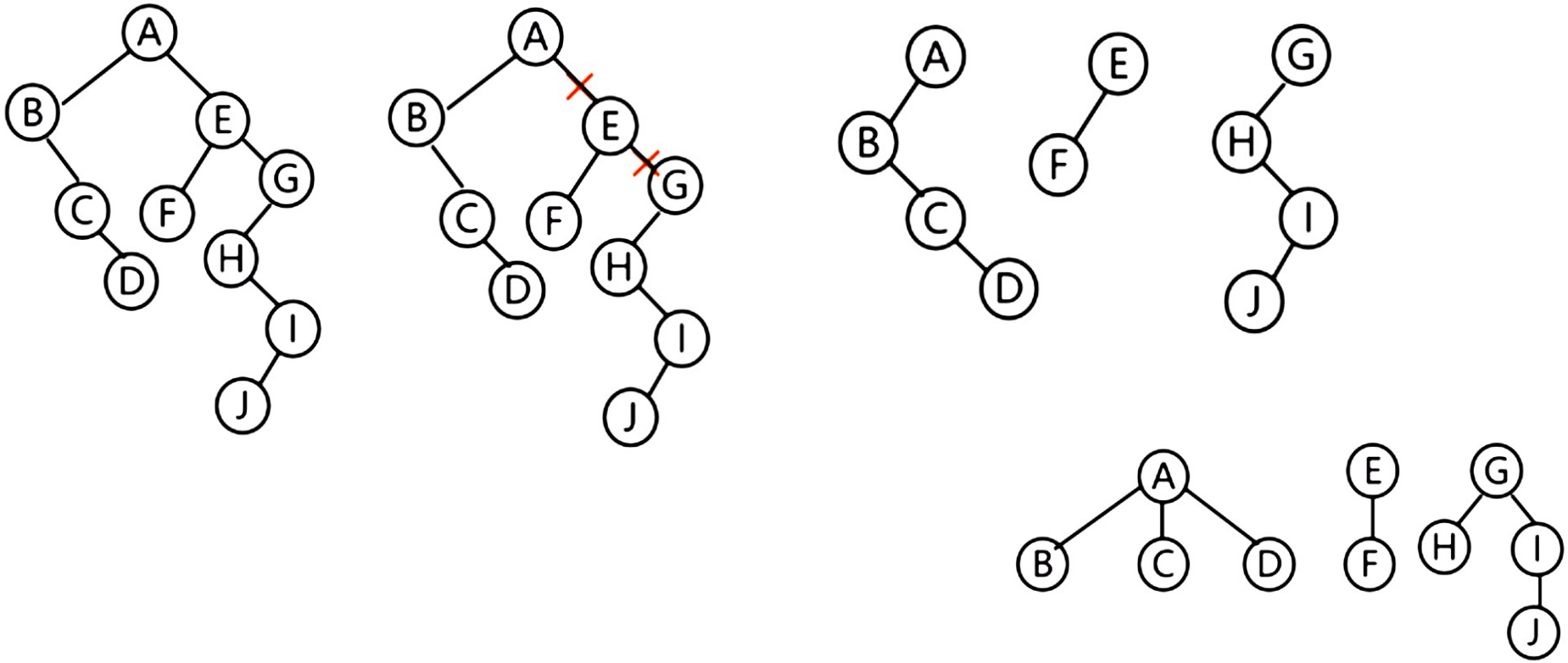


6.5 森林与二叉树的转化

- 二叉树转换成森林

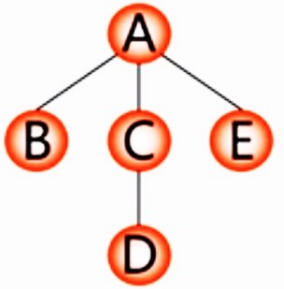
1. 抹线：将二叉树中根结点与其右孩子连线，及沿右分支搜索到的所有右孩子间连线全部抹掉，使之变成孤立的二叉树
2. 还原：将孤立的二叉树还原成树

6.5 例：二叉树转换成森林



6.5 树与森林的遍历

- 树的遍历（三种方式）



- 先根遍历：

  若树不空，则先访问根结点，然后依次先根遍历各棵子树。

- 后根遍历：

  若树不空，则先依次后根遍历各棵子树，然后访问根结点。

- 层次遍历：

  若树不空，则自上而下自左至右访问树中每个结点。

先根遍历： A B C D E

后根遍历： B D C E A

层次遍历： A B C E D

6.5 树与森林的遍历

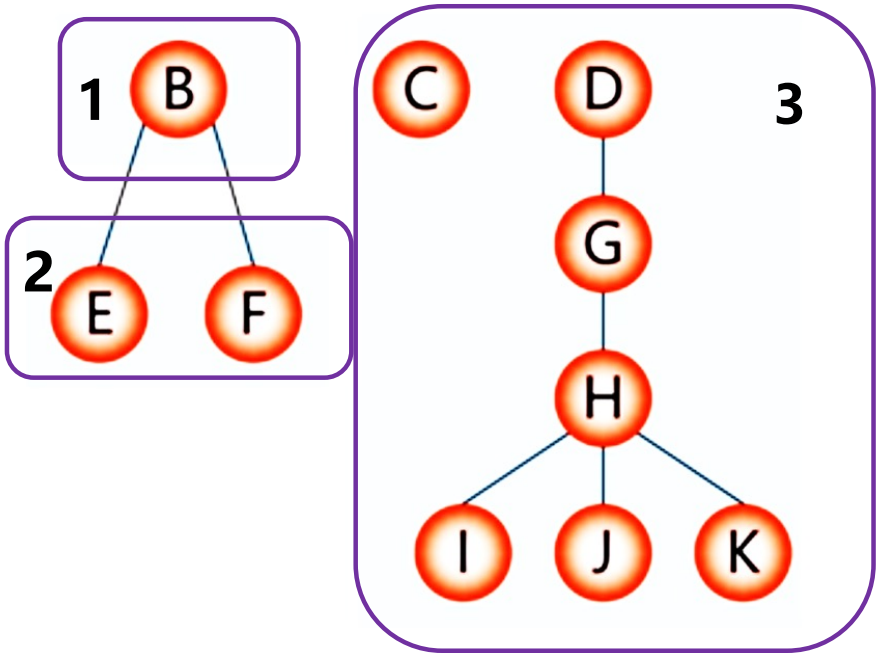
- 森林的遍历

将森林看作由三部分构成：

① 森林中第一棵树的根结点；

② 森林中第一棵树的子树森林；

③ 森林中其他树构成的森林。

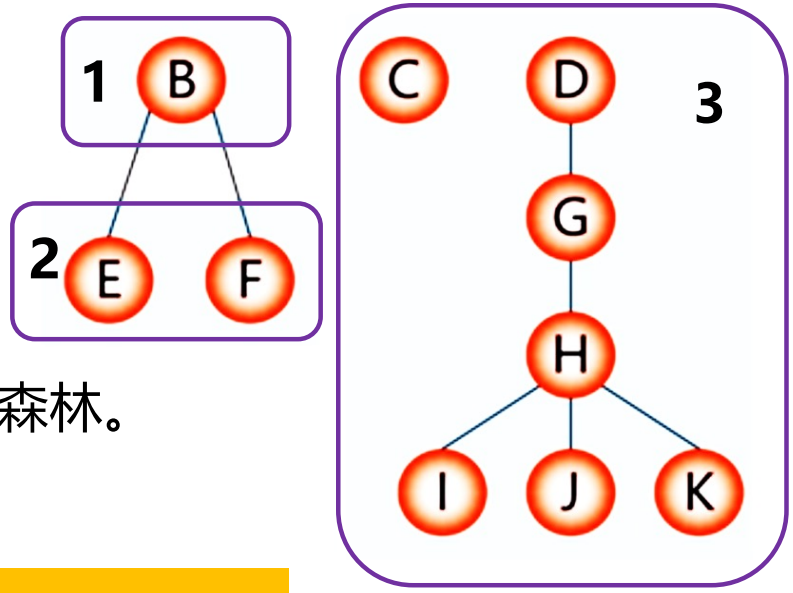


6.5 森林的遍历

- 先序遍历

若森林不空，则

1. 访问森林中第一棵树的根结点；
2. 先序遍历森林中第一棵树的子树森林；
3. 先序遍历森林中（除第一棵树外）其余树构成的森林。



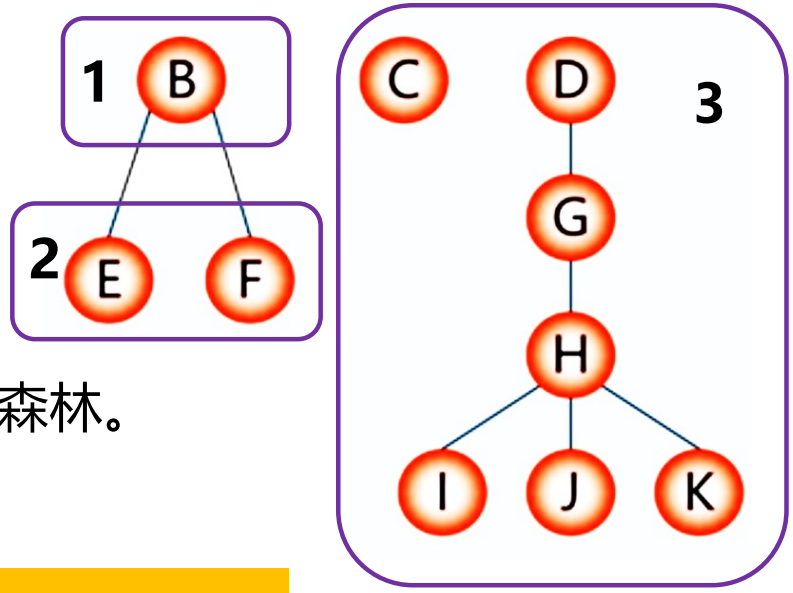
即：依次从左至右对森林中的每一棵树进行先根遍历。

6.5 森林的遍历

- 中序遍历

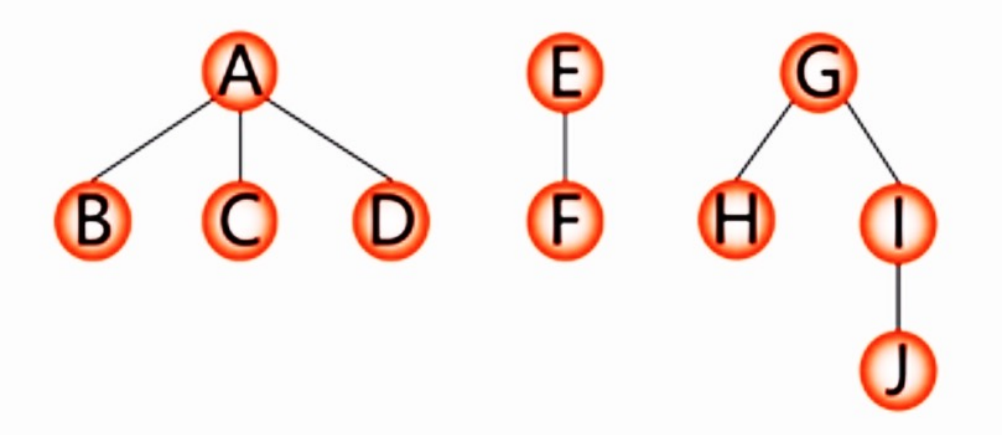
若森林不空，则

1. 中序遍历森林中第一棵树的子树森林；
2. 访问森林中第一棵树的根结点；
3. 中序遍历森林中（除第一棵树外）其余树构成的森林。



即：依次从左至右对森林中的每一棵树进行后根遍历。

例：森林的遍历



遍历结果：

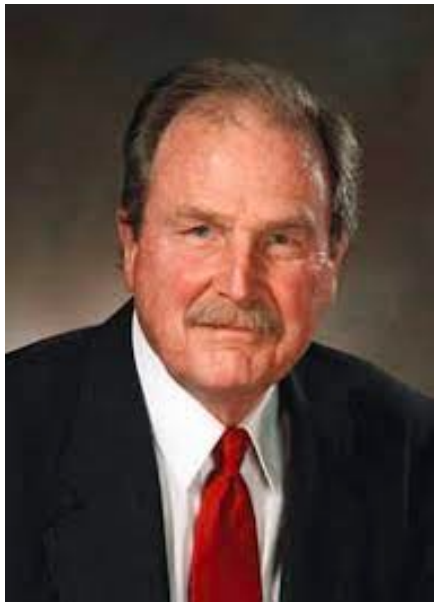
先序遍历： A B C D E F G H I J

中序遍历： B C D A F E H J I G

第6章 树和二叉树

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

6.6 哈夫曼树及其应用



哈夫曼(霍夫曼、赫夫曼) David Albert Huffman(August 9, 1925 - October 7, 1999)。计算机科学的先驱，以他的哈夫曼编码闻名，在他的一生中，对于有限状态自动机，开关电路，异步过程和信号设计有杰出的贡献。

他发明的Huffman编码能够使我们通常的数据传输数量减少到最小。这个编码的发明和这个算法一样十分引人入胜。1950年， Huffman在MIT的信息理论与编码研究生班学习。Robert Fano教授让学生们自己决定是参加期末考试还是做一个大作业。而Huffman选择了后者，原因很简单，因为解决一个大作业可能比期末考试更容易通过。这个大作业促使了Huffman算法的诞生。

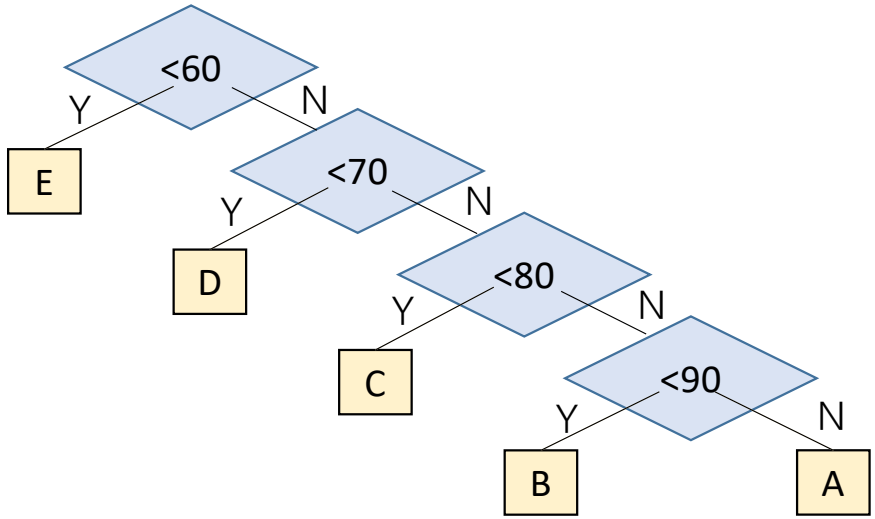
离开MIT后，Huffman来到University of California的计算机系任教，并为此系的学术做出了许多杰出的工作。而他的算法也广泛应用于传真机，图像压缩和计算机安全领域。但是Huffman却从未为此算法申请过专利或其它相关能够为他带来经济利益的东西，他将他全部的精力放在教学上，以他自己的话来说，“我所要带来的就是我的学生。”

6.6.1 哈夫曼树的基本概念

【例】编程：将学生的百分制成绩转换为五分制成绩

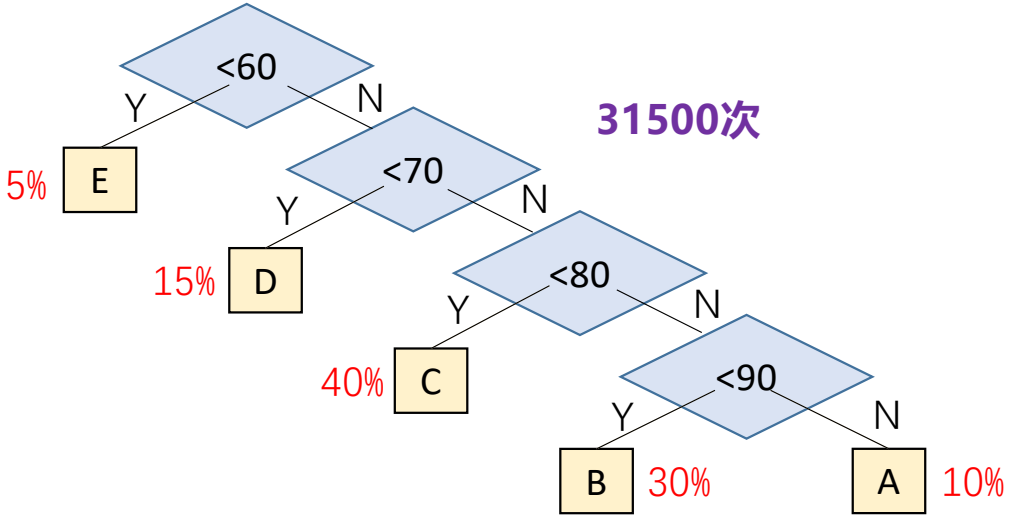
<60: E　　60-69: D　　70-79: C　　80-89: B　　90-100: A

```
if (score< 60)
    grade=='E'
else if (score<70)
    grade =='D'
else if (score<80)
    grade =='C'
else if (score<90)
    grade=='B'
else
    grade =='A'  ;
```



判断树: 用于描述分类过程的二叉树

6.6.1 哈夫曼树的基本概念



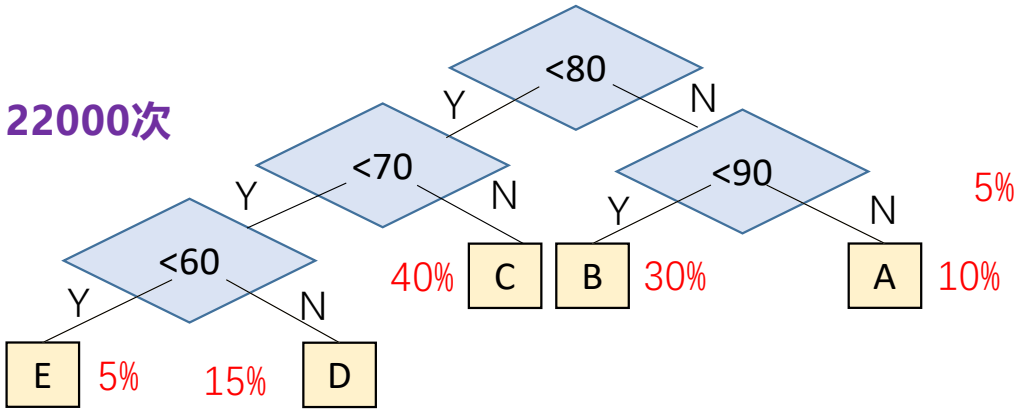
如果每次的输入量很大，则应考虑程序的操作时间。

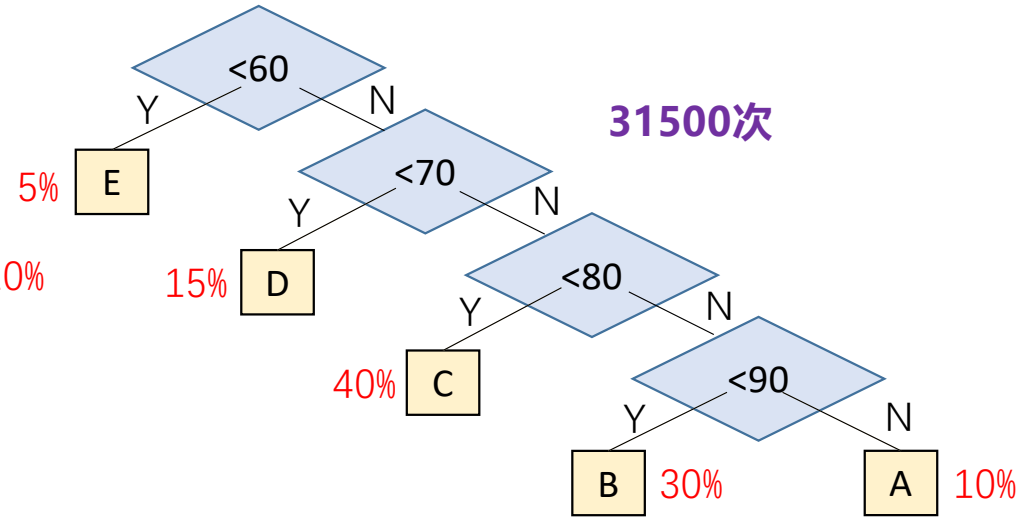
若学生的成绩数据共10000个:

则5%的数据需1次比较，15%的数据需2次比较，40%的数据需3次比较，40%的数据需4次比较。

因此10000个数据比较的次数为: 10000(1x5%+2x15% + 3x40% + 4x40%)= 31500次

6.6.1 哈夫曼树的基本概念

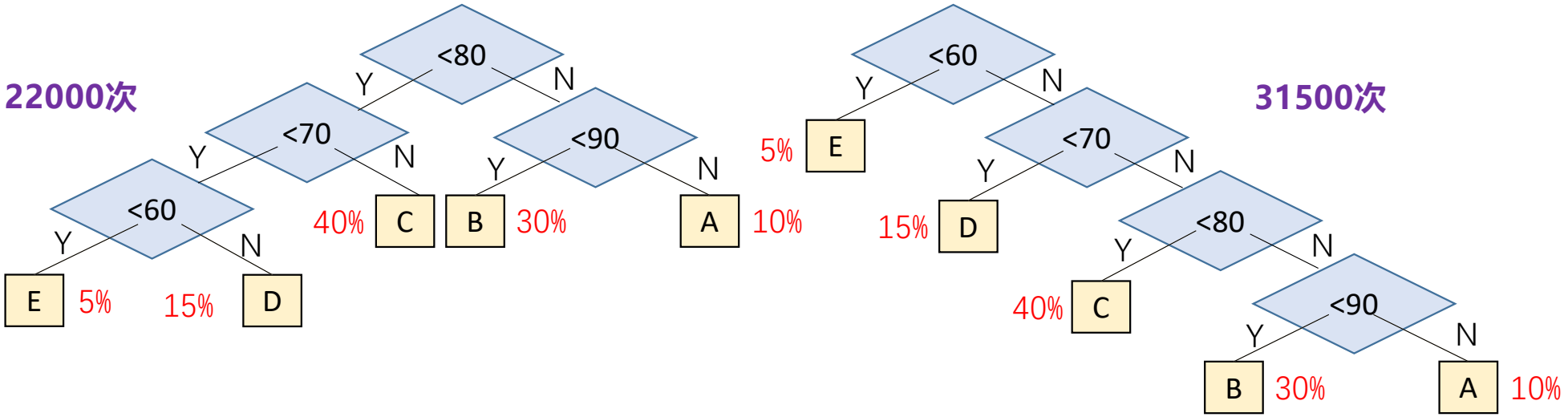




则5%的数据需1次比较，15%的数据需2次比较，40%的数据需3次比较，40%的数据需4次比较，因此10000个数据比较的次数为: 10000 (1x5%+2x15% + 3 x40% +4x40%)= 31500次

10000 (3x 20% + 2x 80%) =22000次)

6.6.1 哈夫曼树的基本概念



显然: 两种判别树的效率是不一样的。

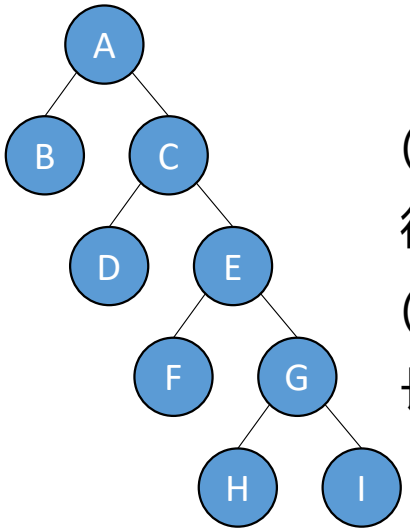
问题: 能不能找到一种效率最高的判别树呢?

哈夫曼树(最优二叉树)

6.6.1 哈夫曼树的基本概念

路径: 从树中一个结点到另一个结点之间的**分支**构成这两个结点间的路径。

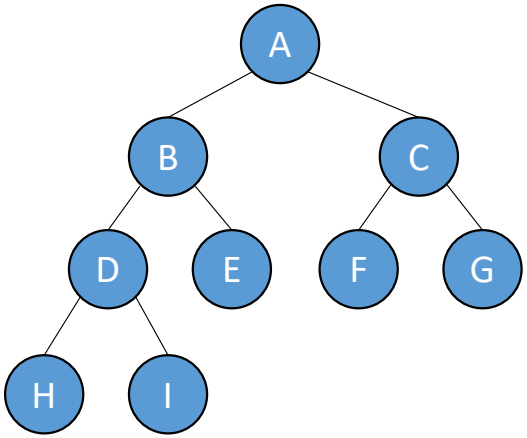
结点的路径长度: 两结点间路径上的**分支数**。



(a)

(a) 从A到B, C, D, E, F, G, H, I的路径长度分别为1, 1, 2, 2, 3, 3, 4, 4。

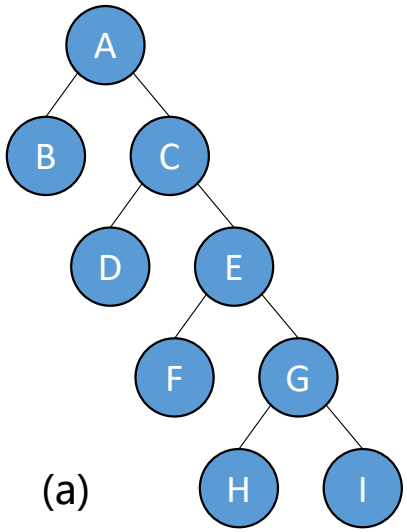
(b)从A到B, C, D, E, F, G, H, I的路径长度分别为1, 1, 2, 2, 2, 2, 3, 3。



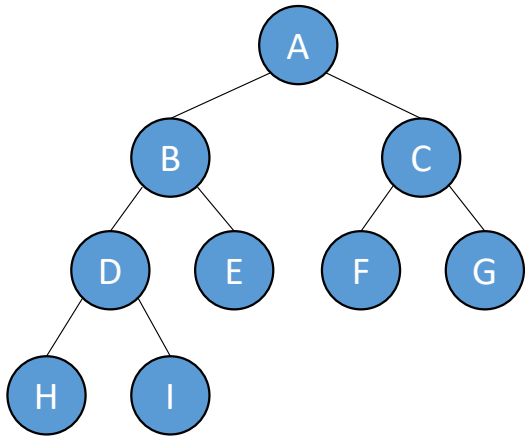
(b)

6.6.1 哈夫曼树的基本概念

树的路径长度: 从树根到每个结点的路径长度之和。记作: TL



(a)



(b)

TL(a) = 0+1+1+2+2+3+3+4+4=20

TL(b) = 0+1+1+2+2+2+2+3+3=16

结点数目相同的二叉树中，完全二叉树是路径长度最短的二叉树。

6.6.1 哈夫曼树的基本概念

权(weight): 将树中结点赋给一个有着某种含义的数值，则这个数值称为该结点的权。

结点的带权路径长度: 从根结点到该结点之间的路径长度与该结点的权的乘积。

树的带权路径长度: 树中所有叶子结点的带权路径长度之和。

记作:

$$WPL = \sum_{k=1}^{n} w_k l_k$$

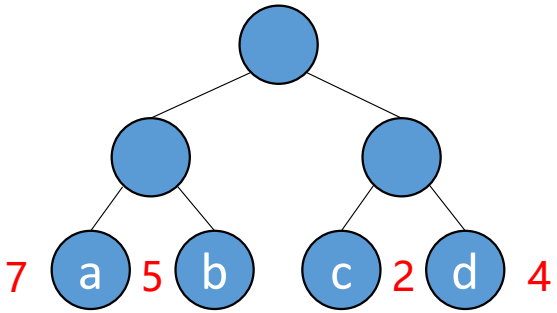
w_k：权值

l_k：根结点到该结点的路径长度

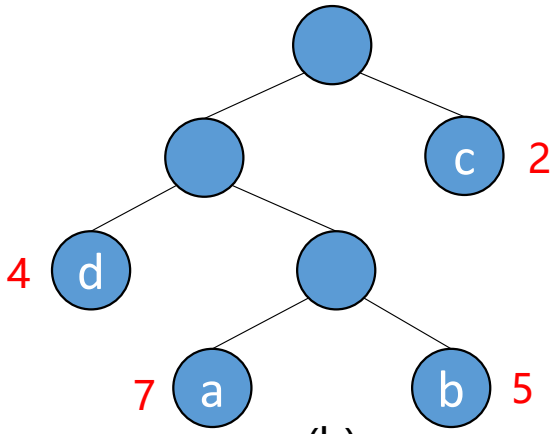
Weighted Path Length

6.6.1 哈夫曼树的基本概念

例: 有4个结点a,b,c,d,权值分别为7,5,2,4,构造以此4个结点为叶子结点的二叉树:



(a)



(b)

带权路径长度是:

(a)WPL=7x2+5x2+2x2+4x2=36

(b)WPL=7x3+5x3+2x1+4x2=46

6.6.1 哈夫曼树的基本概念

哈夫曼树: 最优树

带权路径长度(WPL)最短的树

注

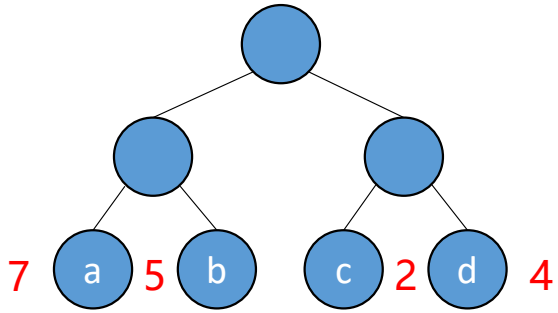
“带权路径长度最短”是在“度相同”的树中比较而得的结果，因此有最优二叉树、最优三叉树之称等等。

哈夫曼树: 最优二叉树

带权路径长度(WPL)最短的二叉树

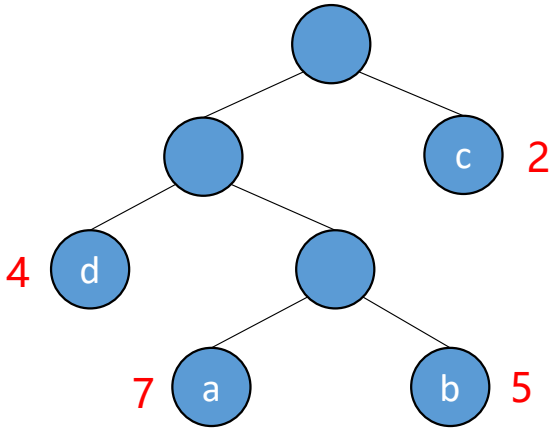
因为构造这种树的算法是由哈夫曼教授于1952年提出的，所以被称为哈夫曼树，相应的算法称为哈夫曼算法。

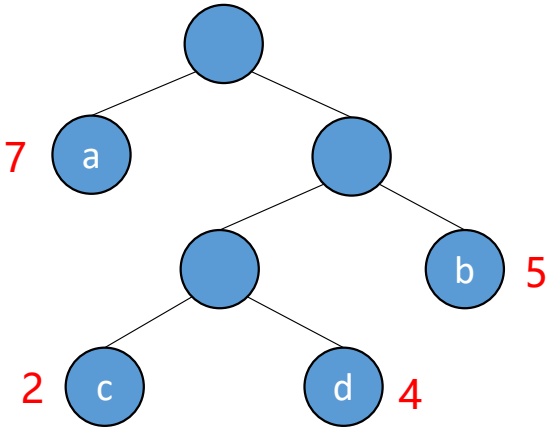
6.6.1 哈夫曼树的基本概念



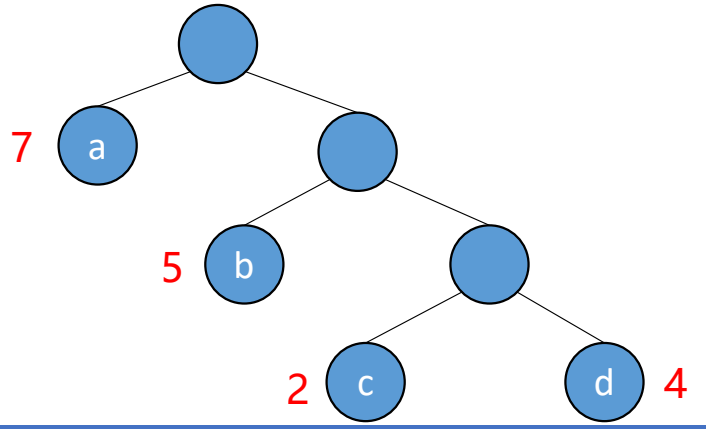
WPL=7x2+5x2+2x2+4x2=36

WPL=7x3+5x3+2x1+4x2=46



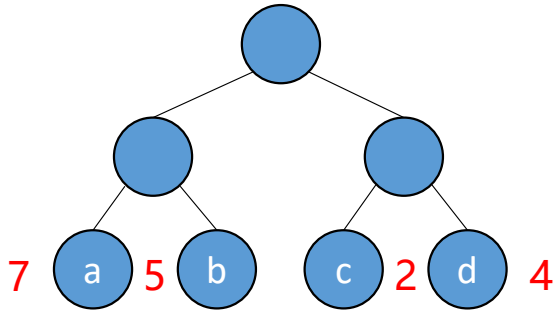


哈夫曼树

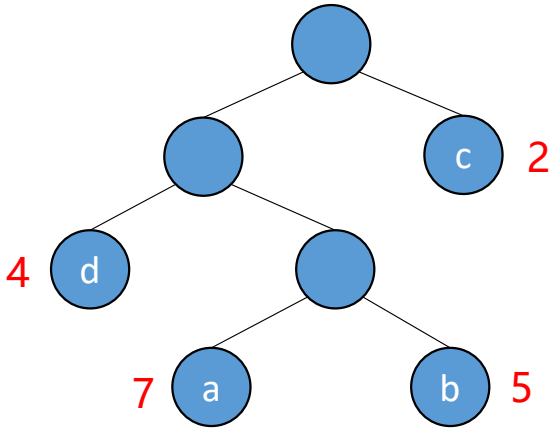


WPL=7x1+5x2+2x3+4x3=35

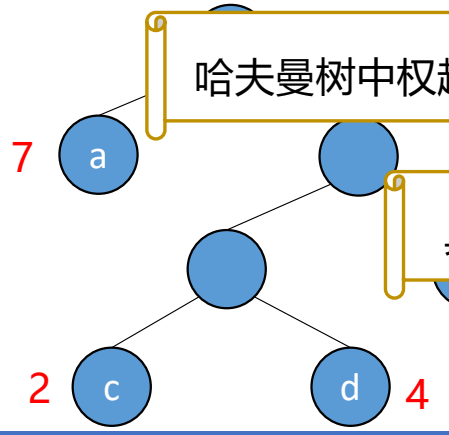
6.6.1 哈夫曼树的基本概念



满二叉树不一定是哈夫曼树

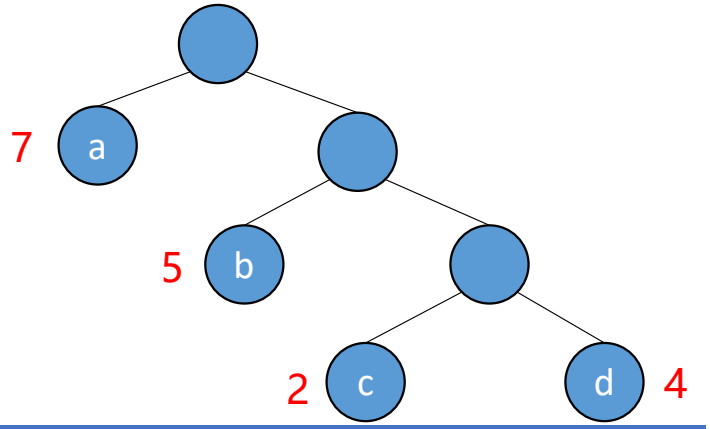


哈夫曼树中权越大的叶子离根越近



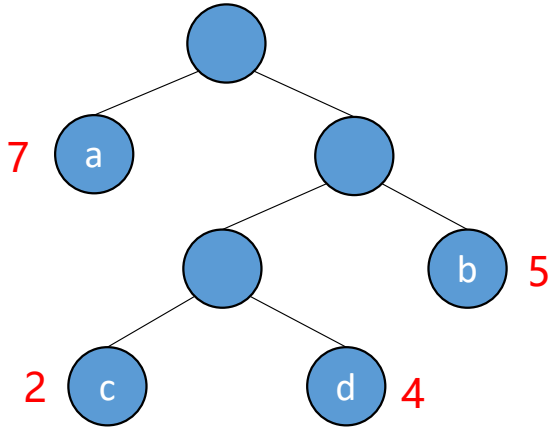
哈夫曼树

具有相同带权结点的哈夫曼树不唯一

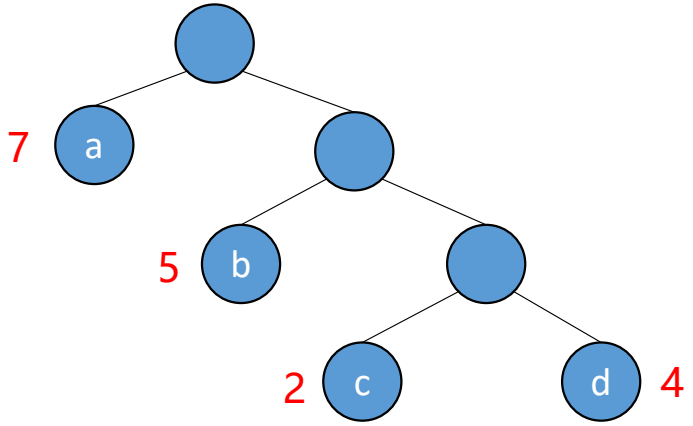


WPL=7x1+5x2+2x3+4x3=35

6.6.2 哈夫曼树的构造算法



哈夫曼树



WPL=7x1+5x2+2x3+4x3=35

WPL=7x1+5x2+2x3+4x3=35

哈夫曼树中权越大的叶子离根越近

贪心算法: 构造哈夫曼树时首先选择权值小的叶子结点

6.6.2 哈夫曼树的构造算法

哈夫曼算法(构造哈夫曼树的方法)

(1) 根据n个给定的权值$\{W_1, W_2, \ldots, W_n\}$构成n棵二叉树的森林$F = \{T_1, T_2, \ldots, T_n\}$，其中$T_i$只有一个带权为$W_i$的根结点。

- 构造森林全是根

(2) 在F中选取两棵根结点的权值最小的树作为左右子树，构造一棵新的二叉树，且设置新的二叉树的根结点的权值为其左右子树上根结点的权值之和。

- 选用两小造新树

(3) 在F中删除这两棵树，同时将新得到的二叉树加入森林中。

- 删除两小添新人

(4) 重复(2)和(3),直到森林中只有一棵树为止，这棵树即为哈夫曼树。

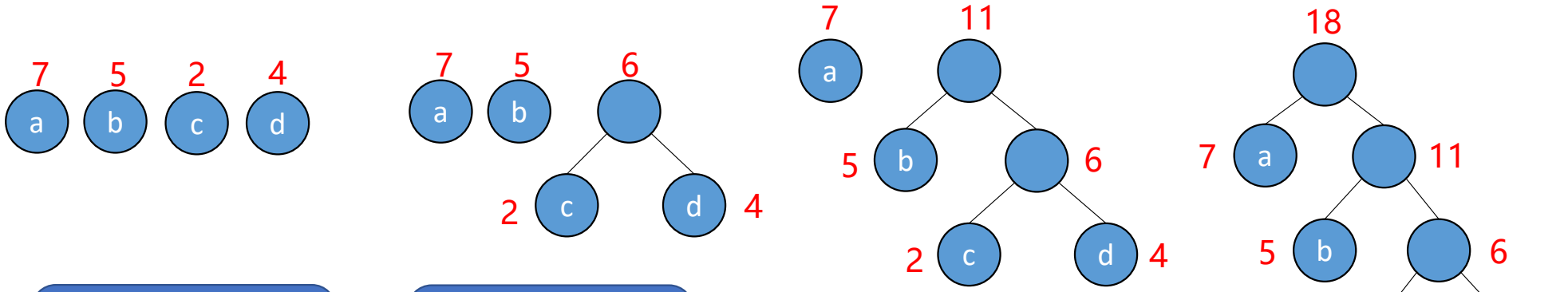
- 重复 2、3剩单根

6.6.2 如何构造哈夫曼树

哈夫曼算法口诀：1、构造森林全是根; 2、选用两小造新树;

3、删除两小添新人; 4、 重复2、3剩单根。

例: 有4个结点a,b,c,d,权值分别为7,5,2,4,构造哈夫曼树。



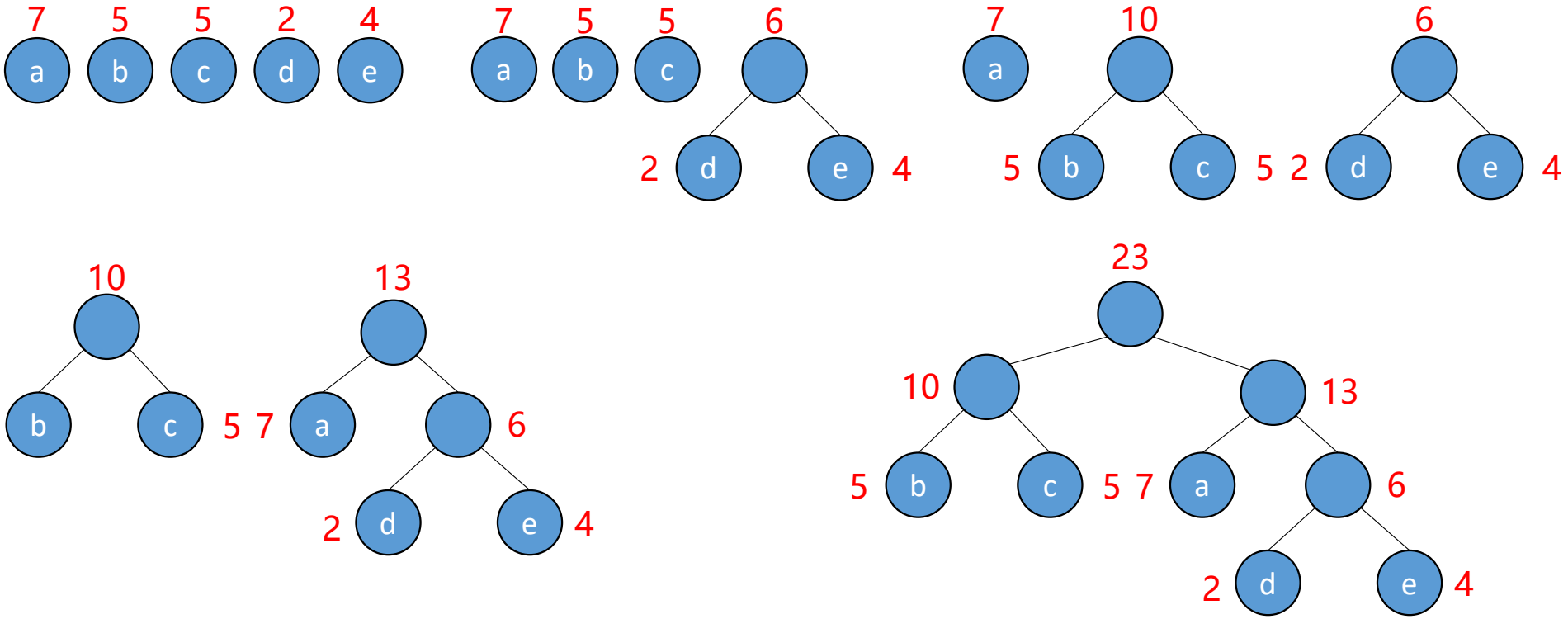
哈夫曼树的结点的度数为0或2，没有度为1的结点。

包含n个叶子结点的哈夫曼树中共有2n- 1个结点。

包含n棵树的森林要经过n-1次合并才能形成哈夫曼树，共产生n-1个新结点

6.6.2 如何构造哈夫曼树

【例】有5个结点a,b,c,d,e,权值分别为7,5,5,2,4,构造哈夫曼树。



总结

1、在哈夫曼算法中，初始时有n棵二叉树，要经过 **n-1** 次合并最终形成哈夫曼树。

2、经过n-1次合并产生n-1个新结点，且这n-1个新结点都是具有两个孩子的分支结点。

可见: 哈夫曼树中共有**n+n-1** = **2n-1**个结点，且其所有的分支结点的度均不为1。

哈夫曼树构造算法的实现

- 采用顺序存储结构——一维结构数组
- 结点类型定义

```
Huffman Tree  H;
```

例如，第1个节点权值为5，即可表示为H[1].weight=5;

```
typedef struct {
        int weight;
        int parent, lch, rch;
}HTNode, *Huffman Tree
```

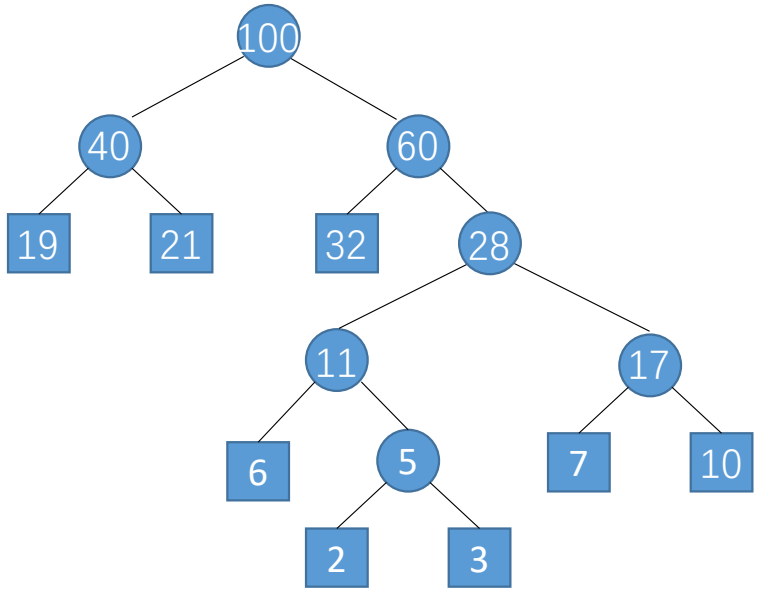
哈夫曼树中结点下标i	weight	parent	lch	rch
1				
2				
3				
4				
……				
……				
2n-1				

哈夫曼树中共有2n-1个节点，不使用0下标，数组大小为2n

	weight		parent	lch	rch
1	7	√	11	0	0
2	19	√	13	0	0
3	2	√	9	0	0
4	6	√	10	0	0
5	32	√	14	0	0
6	3	√	9	0	0
7	21	√	13	0	0
8	10	√	11	0	0
9	5	√	10	3	6
10	11	√	12	4	9
11	17	√	12	1	8
12	28	√	14	10	11
13	40	√	15	2	7
14	60	√	15	5	12
15	100		0	13	14

例，有n=8，权值为W={7,19,2,6,32,3,21,10}

构造哈夫曼树



哈夫曼树构造算法的实现

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

1. 初始化HT[1.....2n-1] : lch=rch=parent=0;
2. 输入初始n个叶子结点: 置HT[1.....n]的weigh值;

哈夫曼树构造算法的实现

【哈夫曼树构造算法】

```
void CreatHuffmanTree (HuffmanTree HT, int n){ //构造哈夫曼树——哈夫曼算法
    if(n<=1) return;
    m=2*n-1;                    //数组共2n-1个元素
    HT=new HTNode[m+1];    //0号单元未用，HT[m]表示根结点
    for(i=1;i<=m;++i) {          //将2n-1个元素的lch、rch、parent置为0
        HT[i].lch=0; HT[i].rch=0; HT[i].parent=0;
    }
    for(i=1;i<=n;++i) cin>>HT[i].weight; //输入前n个元素的weight值
    //初始化结束，下面开始建立哈夫曼树
```

哈夫曼树构造算法的实现

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

1. 初始化HT[1.....2n-1] : lch=rch=parent=0;
2. 输入初始n个叶子结点: 置HT[1.....n]的weight值;
3. 进行以下n-1次合并，依次产生n-1个结点HT[i]，i=n+1.....2n-1:
   a) 在HT[1...i-1]中选两个未被选过(从parent==0的结点中选)的weight最小的两个结点HT[s1]和HT[s2]，s1、s2为两个最小结点下标;
   b) 修改HT[s1]和HT[s2]的parent值: HT[s1].parent=i; HT[s2].parent=i;
   c) 修改新产生的HT[i]:
      - HT[i].weight= HT[s1].weight + HT[s2].weight ;
      - HT[i]. lch=s1; HT[i]. rch=s2;

哈夫曼树构造算法的实现

【续】

```
for( i=n+1;i<=m;i++){     //合并产生n-1个结点——构造Huffman树
        Select(HT, i-1, s1, s2);  //在HT[k](1≤k≤i-1)中选择两个其双亲域为0,
                                  //且权值最小的结点, 并返回它们在HT中的序号s1和s2
        HT[s1].parent=i; HT[s2].parent=i; //表示从F中删除s1,s2
        HT[i].lch=s1; HT[i].rch=s2;           //s1,s2分别作为i的左右孩子
        HT[i].weight=HT[s1].weight + HT[s2] .weight; //i的权值为左右孩子权值之和
    }
}
```

哈夫曼树构造算法的实现

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

例6-2: 设n=8, w={5,29,7,8,14,23,3,11}

试设计Huffman tree

(m=2*8-1=15)

HT的初态

	weight	parent	lch	rch
1	5	0	0	0
.	29	0	0	0
.	7	0	0	0
.	8	0	0	0
.	14	0	0	0
.	23	0	0	0
.	3	0	0	0
8	11	0	0	0
9		0	0	0
.		0	0	0
.		0	0	0
.		0	0	0
.		0	0	0
.		0	0	0
15		0	0	0

哈夫曼树构造算法的实现

构造Huffman tree后，HT为

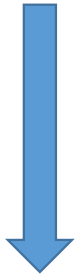
	weight	parent	lch	rch
1	5	9	0	0
.	29	14	0	0
.	7	10	0	0
.	8	10	0	0
.	14	12	0	0
.	23	13	0	0
.	3	9	0	0
8	11	11	0	0
9	8	11	1	7
.	15	12	3	4
.	19	13	8	9
.	29	14	5	10
.	42	15	6	11
.	58	15	2	12
15	100	0	13	14

HT的终态

6.6.3 哈夫曼编码

在远程通讯中，要将待传字符转换成由二进制的字符串:

设要传送的字符为: ABACCDA



若编码为: A——00

B——01

C——10

D——11

00010010101100

若将编码设计为长度不等的二进制编码，即让待传字符串中出现次数较多的字符采用尽可能短的编码，则转换的二进制字符串便可能减少。

6.6.3 哈夫曼编码

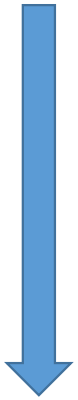
设要传送的字符为: ABACCDA

若编码为: A——0

B——00

C——1

D——01



但: 0000 

0000 11010

AAAA ABA BB

关键: 要设计长度不等的编码，则必须使任一字符的编码都**不是**另一个字符的编码的**前缀**

这种编码称做前缀编码。

6.6.3 哈夫曼编码

问题: 什么样的前缀码能使得电文总长最短?

——**哈夫曼编码**

方法:

1、统计字符集中每个字符在电文中出现的平均概率(概率越大，要求编码越短)。

2、利用哈夫曼树的特点: 权越大的叶子离根越近; 将每个字符的概率值作为权值，构造哈夫曼树。则概率越大的结点， 路径越短。

3、在哈夫曼树的每个分支上标上0或1:

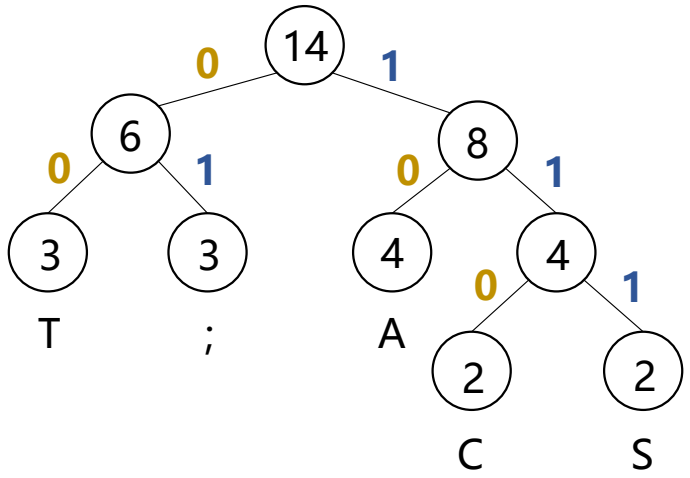
结点的左分支标0，右分支标1

把从根到每个叶子的路径上的标号连接起来，作为该叶子代表的字符的编码。

6.6.3 哈夫曼编码

【例】要传输的字符集 D= {C, A, S, T, ;}

字符出现频率　w={2, 4, 2, 3, 3}



T : 00

; : 01

A : 10

C : 110

S : 111

例：电文是{CAS;CAT;SAT;AT}

其编码是:

11010111011101000011111000011000

反之，若编码是 1101000

则其译文是"CAT"

6.6.3 哈夫曼编码

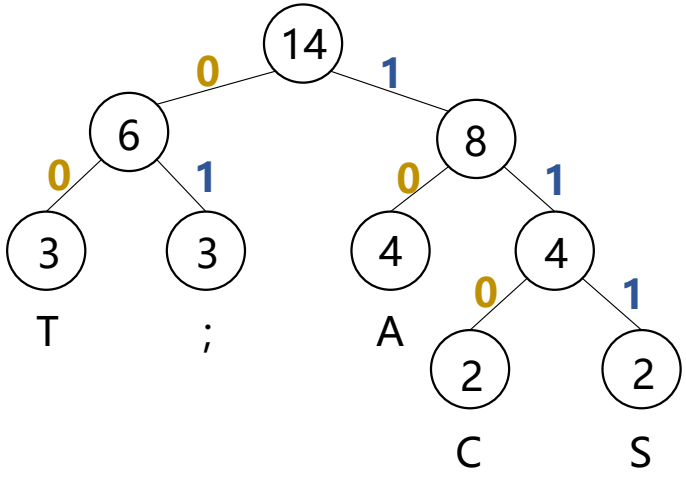
两个问题：

1. 为什么哈夫曼编码能够保证是前缀编码?

因为没有一片树叶是另一片树叶的祖先，所以每个叶结点的编码就不可能是其它叶结点编码的前缀。

2.为什么哈夫曼编码能够保证字符编码总长最短?

因为哈夫曼树的带权路径长度最短，故字符编码的总长最短。



- 性质1 哈夫曼编码是前缀码
- 性质2 哈夫曼编码是最优前缀码

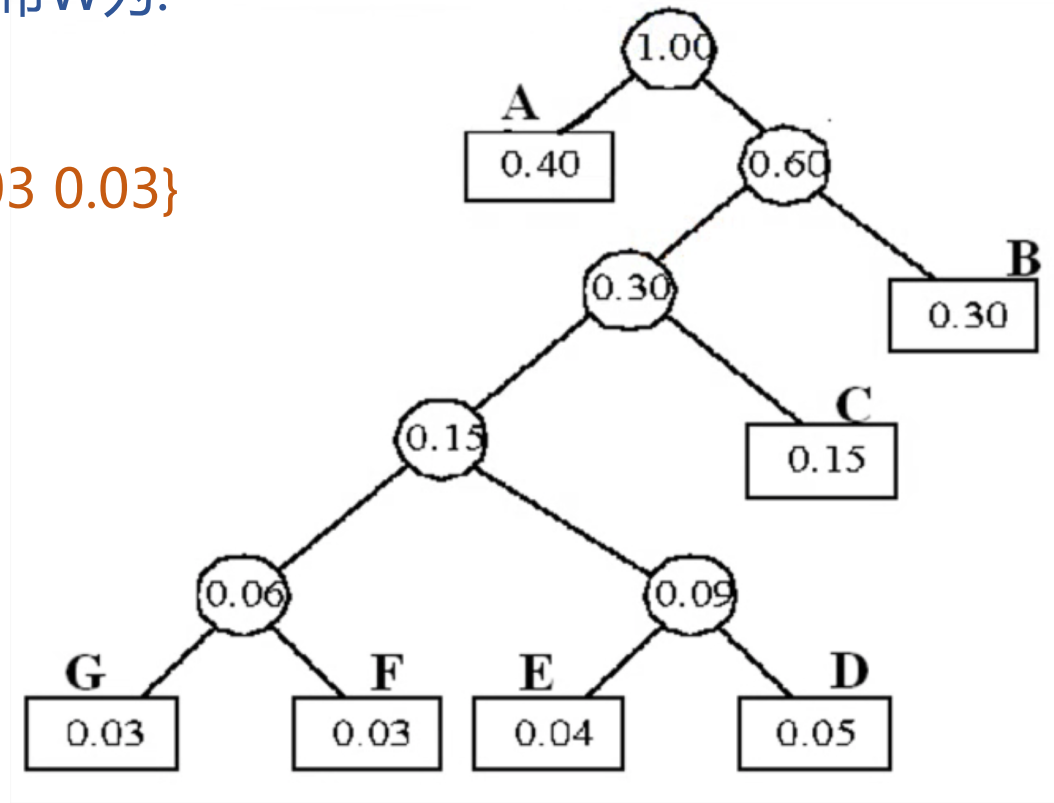
6.6.3 哈夫曼编码

【例】设组成电文的字符集D及其概率分布W为:

D={ A, B, C, D, E, F, G}

W={0.40 0.30 0.15 0.05 0.04 0.03 0.03}

设计哈夫曼编码。

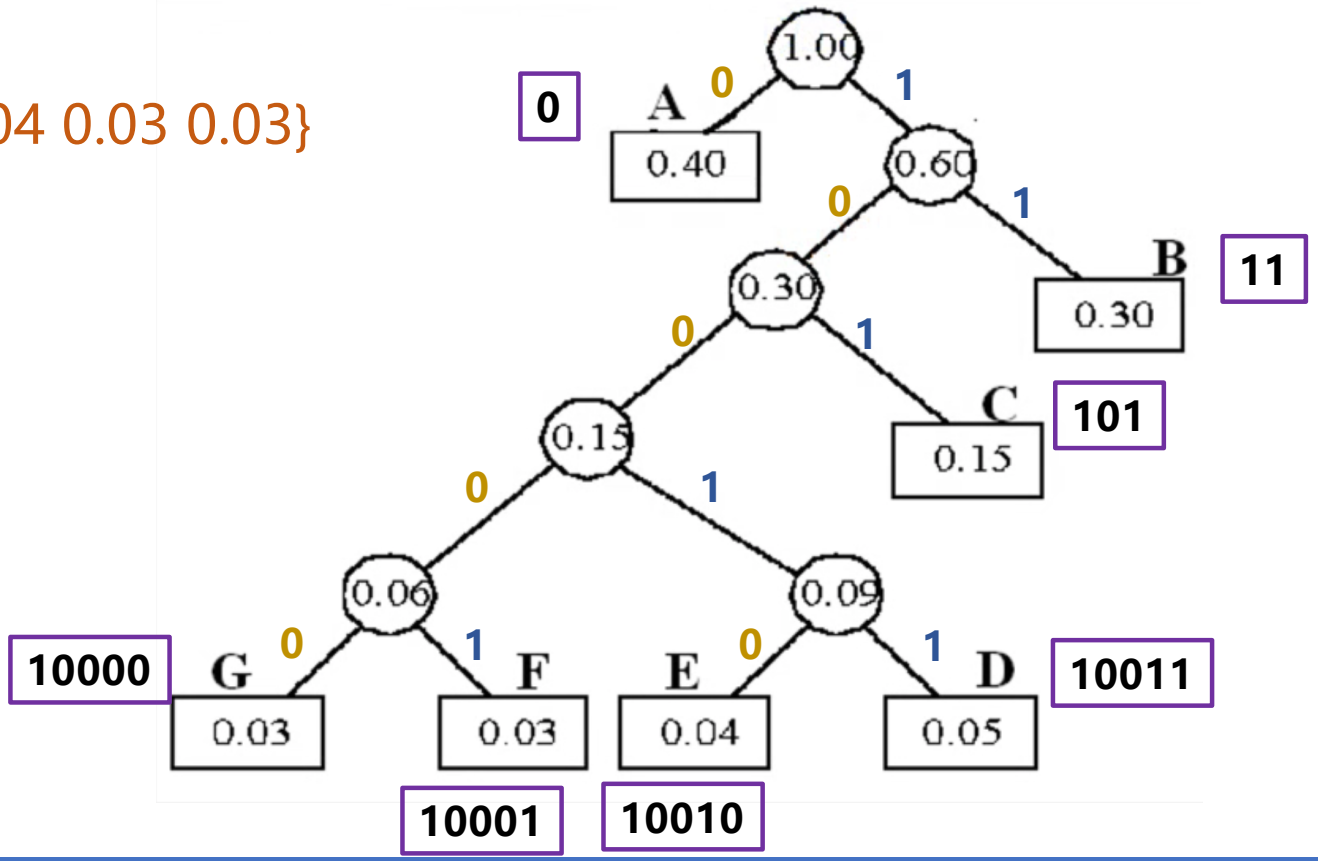


6.6.3 哈夫曼编码

D={ A, B, C, D, E, F, G}

W={0.40 0.30 0.15 0.05 0.04 0.03 0.03}

A	0
B	11
C	101
D	10011
E	10010
F	10001
G	10000



哈夫曼编码的算法实现

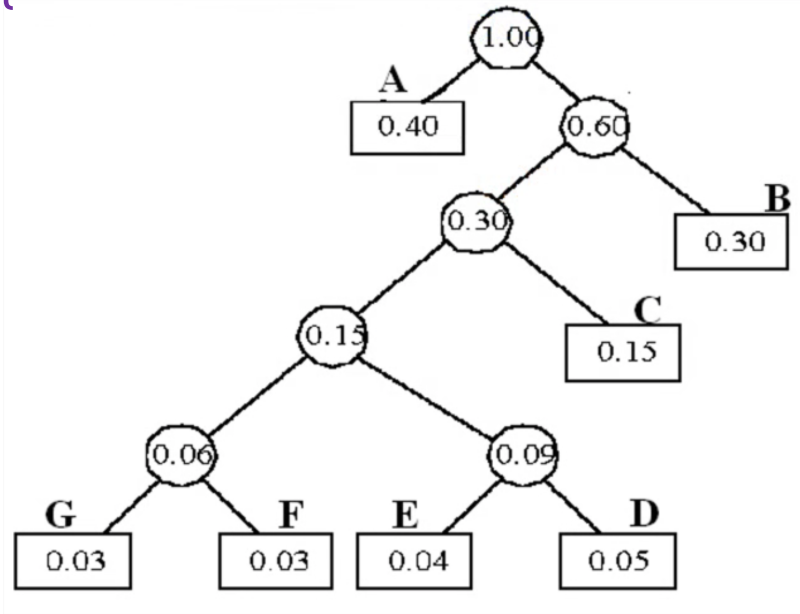
HT[i]

	i	weight	parent	lch	rch
A	1	0.4	13	0	0
B	2	0.3	12	0	0
C	3	0.15	11	0	0
D	4	0.05	9	0	0
E	5	0.04	9	0	0
F	6	0.03	8	0	0
G	7	0.03	8	0	0
	8	0.06	10	7	6
	9	0.09	10	5	4
	10	0.15	11	8	9
	11	0.3	12	10	3
	12	0.6	13	11	2
	13	1	0	1	12

```
for(i=1;i<=n; ++i){
    while(f!=0){

    }
}
```

1 0 0 0 0

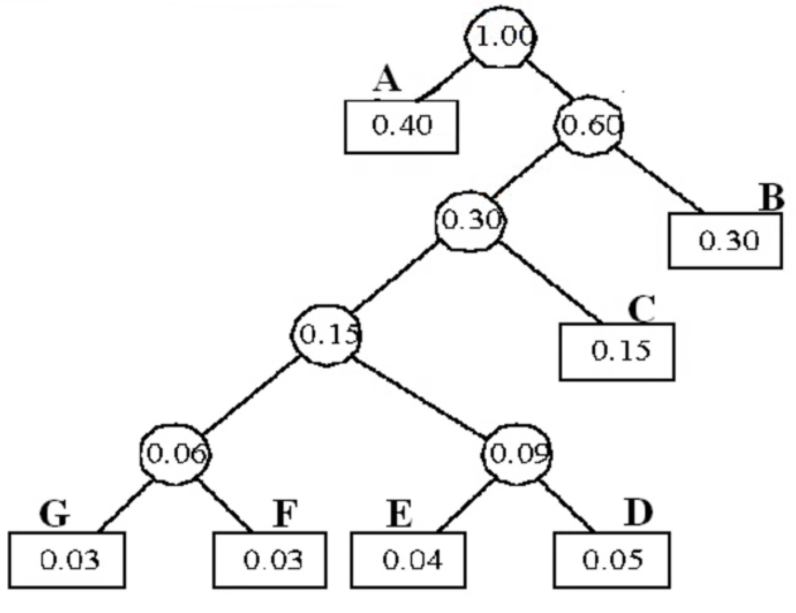


哈夫曼编码的算法实现

HT[i]

	i	weight	parent	lch	rch
A	1	0.4	13	0	0
B	2	0.3	12	0	0
C	3	0.15	11	0	0
D	4	0.05	9	0	0
E	5	0.04	9	0	0
F	6	0.03	8	0	0
G	7	0.03	8	0	0
	8	0.06	10	7	6
	9	0.09	10	5	4
	10	0.15	11	8	9
	11	0.3	12	10	3
	12	0.6	13	11	2
	13	1	0	1	12

	i	HC[i]
A	1	0
B	2	11
C	3	101
D	4	10011
E	5	10010
F	6	10001
G	7	10000



start	0	1	2	3	4	5	6
cd[start]							\0

```
void CreatHuffmanCode(HuffmanTree HT, HuffmanCode &HC, int n){
//从叶子到根逆向求每个字符的哈夫曼编码，存储在编码表HC中
    HC=new char *[n+1];     //分配n个字符编码的头指针向量
    cd=new char [n];          //分配临时存放编码的动态数组空间
    cd[n-1]= ' \0 ' ;         //编码结束符
    for(i=1;i<=n; ++i){      //逐个字符求哈夫曼编码
        start=n-1; c=i; f=HT[i].parent;
        while(f!=0){          //从叶子结点开始向上回溯，直到根结点
            --start;              //回溯一次start向前指一个位置
            if (HT[f].lchild==c) cd[start]= '0' ; //结点c是f的左孩子， 则生成代码0
            else                        cd[start]= '1' ; //结点c是f的右孩子，则生成代码1
            c=f; f=HT[f].parent;                  //继续向上回溯
        }                                                 //求出第i个字符的编码
        HC[i]= new char [n-start];               //为第i个字符串编码分配空间
        strcpy(HC[i], &cd[start]);               //将求得的编码从临时空间cd复制到HC的当前行中
    }
        delete cd;                                    //释放临时空间
} // CreatHuffanCode
```

文件的编码和解码——应用举例

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

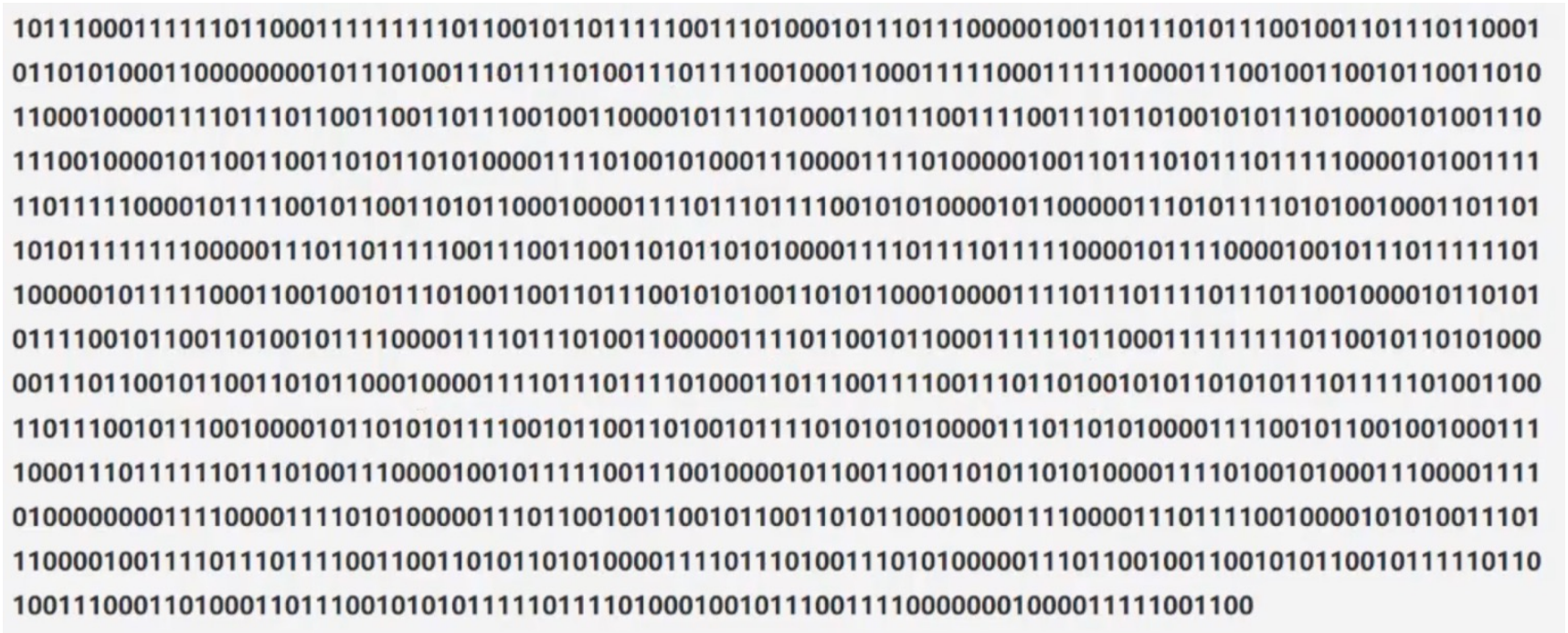
- **明文**

A computer program is a sequence of instructions written to perform a specified task with a computer. The program has an executable form that the computer can use directly to execute the instructions. Computer source code is often written by computer programmers. Source code may be converted into an executable file by a compiler and later executed by a central processing unit.

378个字符，存储ASCII码，每个字符8位，378*8 bit=3024bit

文件的编码和解码——应用举例

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY



1596bit， 1596/3024=52.8%

《数据结构》

文件的编码和解码

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

1. 编码:

① 输入各字符及其权值

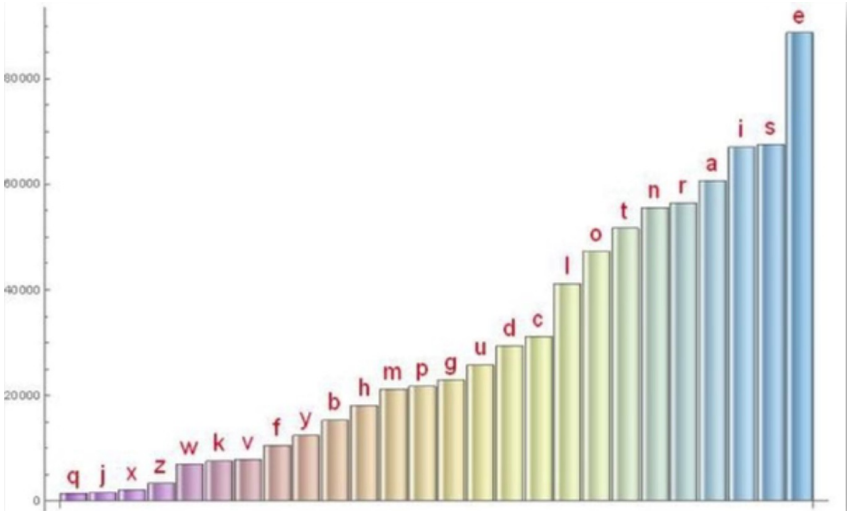
② 构造哈夫曼树——HT[i]

③ 进行哈夫曼编码——HC[i]

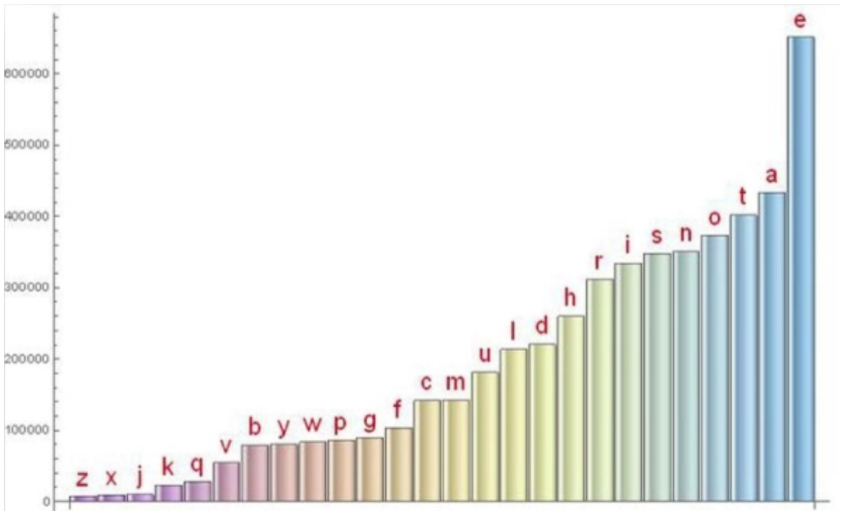
④ 查HC[i]，得到各字符的哈夫曼编码

英语中各字母出现的频率

某研究人员统计了92518个单词各字母频次



某研究人员统计了30篇英语书籍的各字母频次



文件的编码和解码

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

1. 编码:

① 输入各字符及其权值

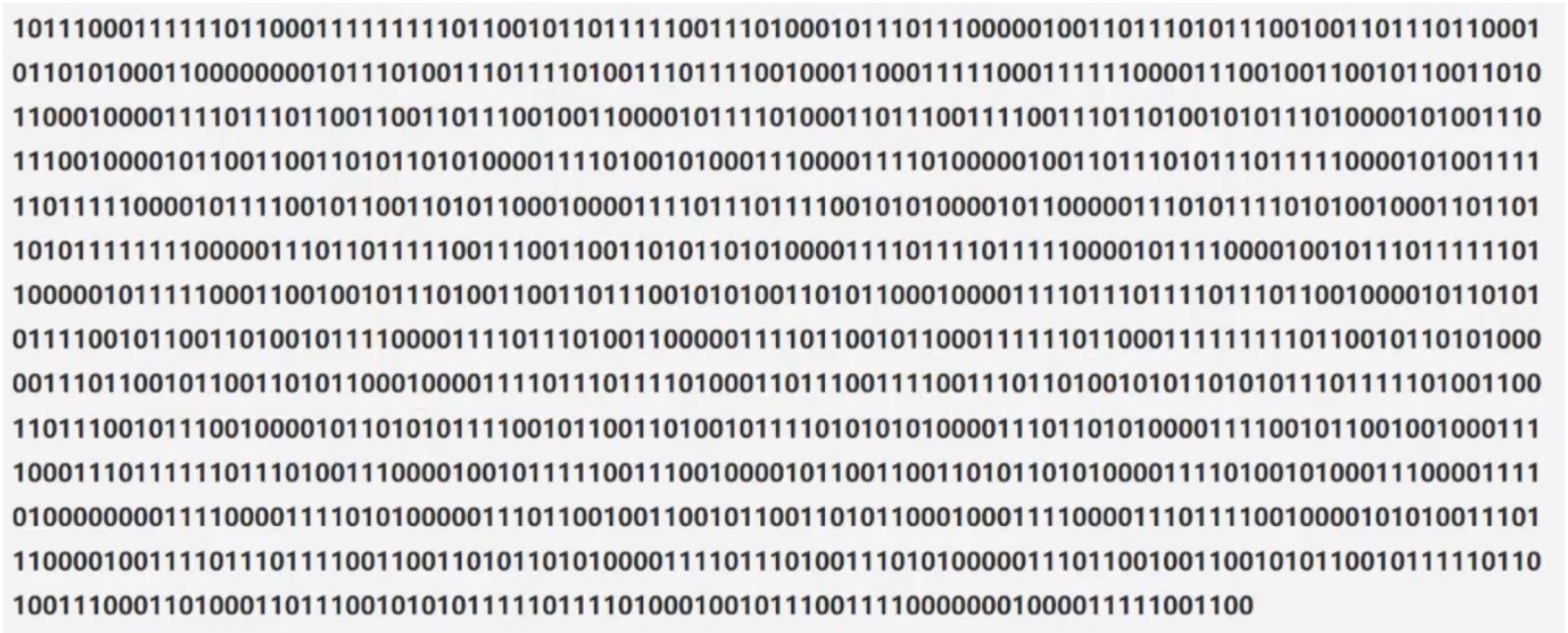
② 构造哈夫曼树——HT[i]

③ 进行哈夫曼编码——HC[i]

④ 查HC[i]，得到各字符的哈夫曼编码

文件的编码和解码——应用举例

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY



1596bit， 1596/3024=52.8%

《数据结构》

文件的编码和解码

2. 解码:

① 构造哈夫曼树

② 依次读入二进制码

③ 读入0，则走向左孩子；读入1,则走向右孩子

④ 一旦到达某叶子时， 即可译出字符

⑤ 然后再从根出发继续译码，直到结束

哈夫曼编码——解码

收到如下编码:

1100011011111010100001111011100101001010011010110101100000

01001001

- 接收字符频度表W

  ((u, 5)，(v, 6)，(w, 2)，(x, 9), (y, 7))

- 构建哈夫曼树HT

- 求出原码报文OC

  uvuwxxxvywyuyyvxxyxxuxuvvyvxy